RADC-TR-90-404, Vol V (of 18)
Final Technical Report
December 1990

DISTRIBUTED PLANNING FOR DYNAMIC ENVIRONMENTS IN THE PRESENCE OF TIME CONSTRAINTS

Northeast Artificial Intelligence Consortium (NAIC)

Susan E. Conry, Robert A. Meyer, Paul R. Cohen,
Victor R. Lesser

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

This effort was funded partially by the Laboratory Director's fund.

Rome Air Development Center
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Services (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-90-404, Volume V (of 18) has been reviewed and is approved for publication.

APPROVED: Northrup Fowler III

NORTHRUP FOWLER III
Project Engineer

APPROVED: Raymond P. Urtz Jr.

RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER: Billy G. Oaks

BILLY G. OAKS
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COES) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1990	Final Sep 84 - Dec 89

4. TITLE AND SUBTITLE
DISTRIBUTED PLANNING FOR DYNAMIC ENVIRONMENTS IN THE PRESENCE OF TIME CONSTRAINTS

5. FUNDING NUMBERS
C - F30602-85-C-0008
PE - 62702F
PR - 5581
TA - 27
WU - 13
(See reverse)

6. AUTHOR(S)
Susan E. Conry, Robert A. Meyer, Paul R. Cohen, Victor R. Lesser

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Northeast Artificial Intelligence Consortium (NAIC)
Science & Technology Center, Rm 2-296
111 College Place, Syracuse University
Syracuse NY 13244-4100

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Air Development Center (COES)
Griffiss AFB NY 13441-5700

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RADC-TR-90-404, Vol V (of 18)

11. SUPPLEMENTARY NOTES (See reverse)
RADC Project Engineer: Northrup Fowler III/COES/(315) 330-7794

This effort was funded partially by the Laboratory Director's fund.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The Northeast Artificial Intelligence Consortium (NAIC) was created by the Air Force Systems Command, Rome Air Development Center, and the Office of Scientific Research. Its purpose was to conduct pertinent research in artificial intelligence and to perform activities ancillary to this research. This report describes progress during the existence of the NAIC on the technical research tasks undertaken at the member universities. The topics covered in general are: versatile expert system for equipment maintenance, distributed AI for communications system control, automatic photointerpretation, time-oriented problem solving, speech understanding systems, knowledge base maintenance, hardware architectures for very large systems, knowledge-based reasoning and planning, and a knowledge acquisition, assistance, and explanation system.

The specific topic for this volume is the real-time simulation of a distributed planning simulation in the context of a dynamic environment.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Artificial Intelligence, Distributed Planning, Plan Recognition, Temporal Reasoning	36
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Block 5 (Cont'd) Funding Numbers

PE - 62702F	PE - 61102F	PE - 61102F	PE - 33126F	PE - 61101F
PR - 5581	PR - 2304	PR - 2304	PR - 2155	PR - LDFP
TA - 27	TA - J5	TA - J5	TA - 02	TA - 27
WU - 23	WU - 01	WU - 15	WU - 10	WU - 01

Block 11 (Cont'd)

This effort was performed as a subcontract by Clarkson University and the University of Massachusetts at Amherst to Syracuse University, Office of Sponsored Programs.

Contents

5.1 Executive Summary

This task is one which started in August of 1988. Our primary goal has been one of developing a testbed environment that is appropriate for research in real time distributed planning. The problem domain selected as an example context in which to investigate the issues of real time distributed planning is forest firefighting. Consequently, much of our activity has been devoted to development of a multiagent firefighting simulation. This was necessary in order to provide an environment in which agents can cooperatively plan to contain fires. In order to accomplish this task, a number of issues related to timing, agent synchronization, management of "thinking time" and "acting time", and agent capabilities have been addressed.

The testbed simulator has not been designed simply as a distribution of an existing centralized firefighting simulator. We found that the issues of time and agent synchronization in a multi-agent environment necessitated a complete redesign of an existing simulation. The new design permits multiple agents to work simultaneously and independently. It includes a facility for defining the characteristics of communications among agents, with available communication media independently specified. In addition, the new simulator reflects a much more realistic terrain representation and a significantly improved fire model than were incorporated in the previous centralized simulator.

The research efforts on this task have been concentrated on formulation and implementation of an appropriate agent model and mechanisms for handling time in general and reasoning strategies for adjudicating allocation of time among various cognitive activities in the planner. It seems clear that time is a critical resource for these types of problems. When time is viewed as a resource, proper allocation of time among subtasks is critical in achieving reasonable performance. One problem central to development of heuristics for determining time allocations is that of formulating ways of handling the fact that time can be viewed in more than one way. It seems evident that the CPU time associated with the planner is measured in seconds to minutes, whereas the time associated with acquiring information regarding the state of the fire may be measured in minutes or hours. Reasoning about actions in an environment such as this requires that the planner understand and be able to deal with these extreme differences in scale. Effectively, there are two types of time: "execution time" and "action time" or "internal time" and "external time". These two types of time share some attributes, but are fundamentally different in others (as far as the planner is concerned). Research concerning a model of distributed planning, agent characteristics, and effective ways of modeling time has been initiated and mechanisms for experimenting with time (as perceived by the agents in the system) have been incorporated in the simulator design. In addition, qualitative reasoning

about time allocation has been investigated and appropriate algorithms have been implemented. Finally, an investigation of ways in which multiple "firebosses" can be accommodated has been initiated.

5.2 Introduction

As has been mentioned, we utilize the firefighting domain as an example of a real time planning problem in which time is a valuable resource that must be carefully allocated. It seems evident that in any real time planning environment, it is imperative that planning and situation assessment be handled in an appropriate manner. One approach to the problem involves a system architecture that incorporates a planning model that interleaves planning and monitoring tasks. The planner cannot achieve reasonable performance levels without relying on a situation assessment "agent" to provide timely information. Likewise, the situation assessment task is driven, in part, by the planner. Computation time must be shared among the agents in an appropriate manner.

In the firefighting domain, time can be viewed from two perspectives. Computation time is utilized by the cognitive agents in determining what actions should be taken to further their goals. Execution time is required to accomplish various tasks in pursuit of those goals. Time management becomes a problem of managing both computation time and execution time in such a way that each module is allocated time "proportional" to its current ability to further the firefighting effort.

In the work that has been done, our attention has been focused on development of a distributed firefighting simulator, on a design for an agent, on development of strategies for sharing computational resources among cognitive agents in a dynamically changing environment, and on distributing the fire controller's function. The work discussed in this document should be viewed as a reflection of preliminary design efforts. It is not yet mature. As this work does mature, it is anticipated that many of these ideas may undergo significant revision.

In the sections that follow, we discuss our distributed firefighting system's structure, a preliminary design for the cognitive component of an agent, various aspects of managing time in a real time environment, and factors that must be addressed when top level control is distributed.

5.3 Phoenix System Structure

To facilitate experiments our firefighting system, Phoenix, is built in four layers. The lowest is a task coordinator that maintains the illusion of simultaneity among many cognitive, perceptual, reflexive and environmental processes, on a serial machine.

The next layer implements the Phoenix environment itself—the maps of Yellowstone National Park, and the simulations of fires. The third layer contains the definitions of the components of agents—our specific agent design. The fourth layer describes the current organization of agents, their communication and authority relationships.

5.3.1 The Phoenix Environment, Layers 1 and 2

The two lowest layers in Phoenix, called the task coordinator layer and map layer, respectively, comprise the Phoenix discrete event simulator. We discuss the task coordinator first. It is responsible for the illusion of simultaneity among the following events and actions:

- **Fires:** Multiple fires can burn simultaneously in Phoenix. Fires are essentially cellular automata that spread according to local environmental conditions, including wind speed and direction, fuel type, humidity, and terrain gradient.
- **Agents' physical actions:** Agents move from one place to another, report what they perceive, and cut fireline.
- **Agents' "internal" actions:** Internal actions include sensing, planning, and reflexive reactions to immediate environmental conditions.

These tasks are not generated at the task coordinator level of Phoenix, just scheduled on the cpu there. Fire tasks are generated at the map layer, and agent tasks are generated at the levels described in subsequent sections.

Typically, the task coordinator manages the physical and internal actions of several agents (e.g., one fireboss, four bulldozers, and a couple of watchtowers), and one or more fires. The illusion of continuous, parallel activity on a serial machine is maintained by segregating each process and agent activity into a separate task and executing them in small, discrete time quanta, ensuring that no task ever gets too far ahead of or behind the others. The default setting of the synchronization quantum is five minutes, so all tasks are kept synchronized to within five minutes of each other. The quantum can be increased, which improves the cpu utilization of tasks and makes the testbed run faster, but this increases the simulation-time disparity between tasks, magnifying coordination problems such as communication and knowing the exact state of the world at a particular time. Conversely, decreasing the quantum reduces how 'out of synch" processes can be, but increases the running time of the simulation.

The task coordinator manages two types of time: cpu time and simulation time. CPU time refers to the length of time that processes run on a processor. Simulation time refers to the "time of day" in the simulated environment. Within the predefined time quantum, all simulated parallel processes begin or end at roughly the same

simulation time. To exert real-time pressure on the Phoenix planner, every cpu second of "thinking" is followed by K simulation-time minutes of activity in the Phoenix environment. Currently K = 5, but this parameter can be modified to experiment with how the Phoenix planner copes with different degrees of time pressure.

The fire simulator resides at Phoenix's map layer; that is, the map layer generates tasks that, when executed by the task coordinator, produce dynamic forest fires. Phoenix's map, which represents Yellowstone National Park, is a composite of several two dimensional structures, and stores information for each coordinate about ground-cover, elevation, features (roads, rivers, houses, etc.), and fire-state. The fire itself is implemented as a cellular automaton in which each cell at the boundary decides whether to spread to its neighbors, depending on the local conditions just mentioned and global conditions such as wind speed and direction (currently, we do not model local variations in weather conditions). These conditions also determine the probability that the fire will jump fireline and natural boundaries.

The Phoenix discrete event simulation is generic. It can manage any simulations that involve maps and processes. For example, we could replace the forest fire environment with an oil-spill environment. We could replace our map of Yellowstone with oceanographic maps of, say, Prince William Sound. Fire processes have spatial extent, and spread according to wind speed, direction, fuel type, terrain, and so on. They could easily be replaced with oil-slick processes, which also have spatial extent, and spread according to other rules. Similarly, we could replace the definitions of bulldozers and airplanes with definitions of boats and booms.

5.3.2 Agent Design, Layer 3

The third layer of Phoenix is our specific agent design, which is constrained by the forest fire environment. Because events happen at two dramatically different time scales, we designed an agent with two parallel and nearly-independent mechanisms for generating actions. One generates reflexive actions very quickly—on the order of a few seconds of simulated time—and the other generates plans that may take hours of simulated time to execute. This longer-term planning can be computationally intensive, because it incurs a heavy time penalty for switching contexts when interrupted. For this reason, the cognitive component is designed to do only one thing at a time (unlike sensors, effectors, or reflexes, where multiple activities execute in parallel). Both the cognitive and reflexive component have access to sensors, and both control effectors, as shown in Figure 1.

The agent interacts with its environment through its sensors and effectors, and action is mediated by both the reflexive and the cognitive components. Sensory information may be provided autonomously or may be requested, and sensors' sensitivity may be adjusted by the cognitive component. Effectors produce actions in the world

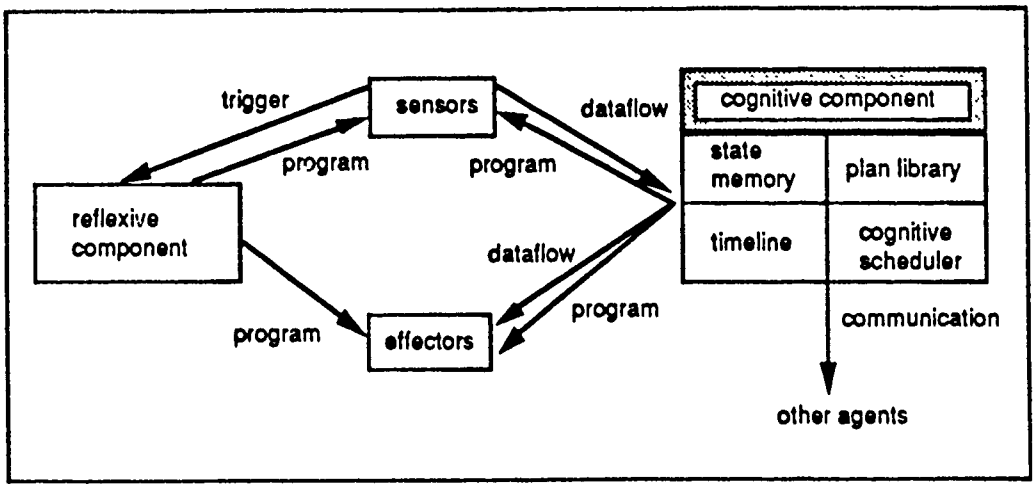

Figure 1: Phoenix agent design

such as information gathering, building fireline, and moving.

Reflexes are triggered by output of sensors. They change the programming of effectors to prevent catastrophes, or they fine tune the operation of effectors. For example, a bulldozer is stopped by a reflex if it is about to move into the fire, and reflexes handle the fine tuning necessary for the bulldozer to follow a road. Reflexes are allotted almost no cpu time, and have no memory of events, so they cannot produce coordinated sequences of actions. They are designed for rapid, unthinking action. Although some researchers have suggested that longer-term plans can emerge from compositions of reflexes [1, 2], we do not believe that compositions of reflexes can handle temporally-extensive planning tasks such as resource management, or spatially-extensive tasks such as path planning with rendezvous points for several agents. Thus, we have adopted a design in which reflexes handle immediate tasks and a cognitive component handles everything else.

The cognitive component of an agent is responsible for generating and executing plans. Instead of generating plans de novo, as classical hierarchical planners did, the Phoenix cognitive component instantiates and executes stored skeletal plans. We believe this is a good design for the forest fire environment because, first, a relatively small number of skeletal plans is probably sufficient to cope with a wide range of fires; and, second, the store/recompute tradeoff suggests relying on stored plans, rather than computing them, in real-time situations. In addition to controlling sensors and effectors, the cognitive component handles communications with other agents (including integrating sensor reports), and it responds to flags set when reflexes exe-

cute. It also engages in a wide range of "internal" actions, including projection (e.g., where will the fire be in 20 minutes?), plan selection and scheduling, plan monitoring, error recovery, and replanning. Our implementations of some of these capabilities are quite rudimentary, and leave much room for improvement.

In overview, this is how the cognitive component works: in response to a situation such as a new fire, an appropriate plan is retrieved from the *plan library* and placed on the *timeline*. *State memory* stores information, such as weather, resource conditions, and sensory input, that helps the cognitive agent select appropriate plans and instantiate the variables of the chosen plan for the current situation. For example, if the fire is small and nearby, and the weather is calm, then a one-bulldozer plan will be retrieved and instantiated with situation-specific information such as the wind speed and the current location of the fire. The actions in a plan are eventually selected for execution by the *cognitive scheduler*, to be described shortly. At any time during this process, sensory data may trigger reflexive actions. For example, if the cognitive component is executing a command to move to a destination, and a sensor reports fire ahead, then the reflexive component will send a command to reverse direction. This happens very fast relative to the cycle time of the cognitive component, so the reflexive component sets a flag to tell the cognitive component what it did. When the cognitive component notices the flag, it might modify its plan. The analogy here is to our own reflexes, which yank us away from hot surfaces long before our cognitive apparatus becomes aware of the problem.

With this overview in mind, we consider the operation of the cognitive component in detail. We will focus on the operation of the *fireboss* agent, which plans the activities of other agents such as bulldozers and crews. Each of these, in turn, plans how to carry out the directives of the fireboss. Because bulldozers and crews have the same architecture as the fireboss (namely that shown in Figure 1), they can reason in exactly the same way. In the following discussion, we first describe planning when things go according to plan, and then describe error handling, interruptions, and other unexpected events.

When a fire is reported, an *action* called "deal with fire" is retrieved from the plan library and used to create a *timeline entry*, in this case called "deal with fire 27", which is added to the timeline (Figure 2). Actions are general representations of the cognitive activities the agent can perform, such as path planning or communication, and describe applicability conditions, resource constraints and uninstantiated variables. Creating a timeline entry instantiates an action: binding its variables and adding the temporal constraints that relate it to other actions the agent has chosen to execute. Although timeline entries represent actions, it is not quite accurate to say they are executed (although we will use this terminology where the accurate description is too awkward). In fact, when a timeline entry is created, it inherits a set of *execution methods* from the action it instantiates. Each of these methods will execute

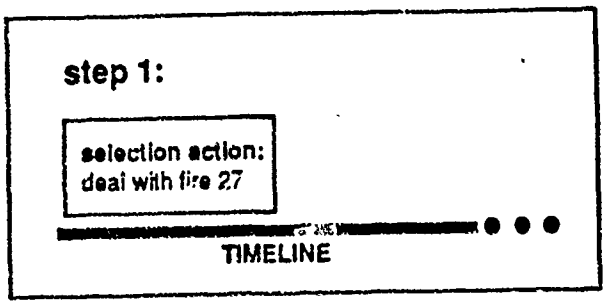

Figure 2: Contents of fireboss's timeline after being notified of a new fire: action to search for a plan to deal with the fire

the desired action; they differ along dimensions such as the time they require and the quality of their outputs. For example, a single action "plan a path" points to several path-planning algorithms, some which run quickly and return adequate paths, and some that run longer but produce shorter paths. When a timeline entry is selected for execution, the execution method most appropriate to the current circumstances is chosen. By delaying the choice of methods, the cognitive scheduler can reason about its own use of time, and select execution methods that are suited to emerging time constraints.

If there are entries on the timeline (e.g., "deal with fire 27") then the cognitive scheduler of the Phoenix cognitive component currently makes three decisions:

- Which action to execute next
- How much time is available for its execution
- What execution method should implement the action

The cognitive scheduler always selects the "next" action on the timeline to execute, but often, several actions have this distinction and a choice must be made. Actions on the timeline may be unordered (and thus equally entitled to "go first") for several reasons: skeletal plans often leave actions unordered so that the cognitive scheduler has flexibility at execution time to select the best order. Or, frequently, the agent is executing several plans simultaneously. This happens, for example, when several fires are reported. The planner formulates plans for each, but doesn't specify temporal constraints among actions from different plans. In the current example, however, the only action on the timeline is "deal with fire 27," so the cognitive scheduler determines

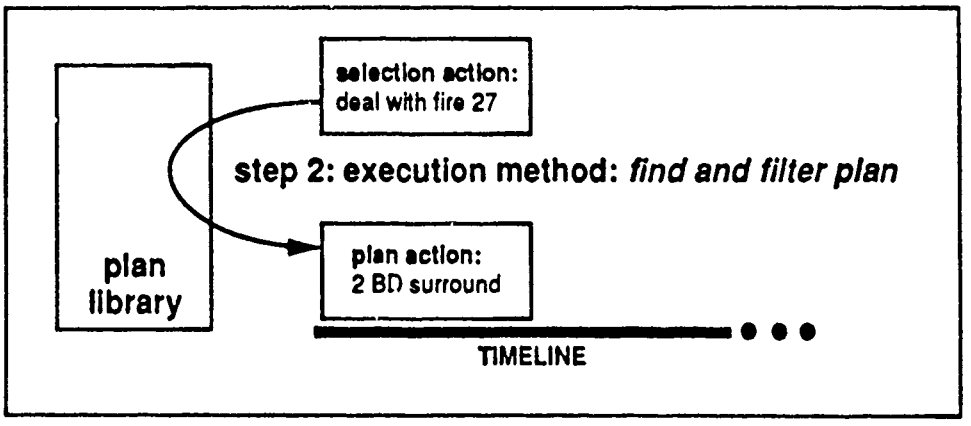

Figure 3: The fireboss executes timeline action, deal with fire 27, which searches the plan library, selects the 2 BD surround plan as appropriate for dealing with new fire, and places the new plan on the timeline

how much time is available to execute it and selects an execution method. In this case, it selects a method called find and filter plan (step 2, Figure 3). Its effect, when executed, is to search the plan library for a plan to "deal with fire 27." First it finds all plans for dealing with fires of this type, then it filters the infeasible ones, then selects from the candidates to find the most appropriate one, and lastly, it adds a new action to the timeline called "2 BD surround." (This plan involves sending two bulldozers to a rendezvous point, then to the fire, after which they cut fireline in opposite directions around the fire.)

Once again, the cognitive scheduler selects an action (the only one is "2 BD surround") assesses how much time is available, and selects an execution method. In this case, the method is *expand* plan. The result is to add a network of actions, partially ordered over time, to the timeline (step 3, Figure 4). The network starts with a placeholder action, s, followed by two unordered actions that allocate bulldozers 1 and 2, respectively. The next action determines the rendezvous point for the bulldozers. Then two unordered actions bind the variables in the plan with the current wind direction and the previously-determined rendezvous point. Space precludes showing the rest of the plan in Figure 4.

The cognitive scheduler again looks at the timeline, and now must make a decision about which action to select. The "allocate bulldozer" actions are unordered, so one

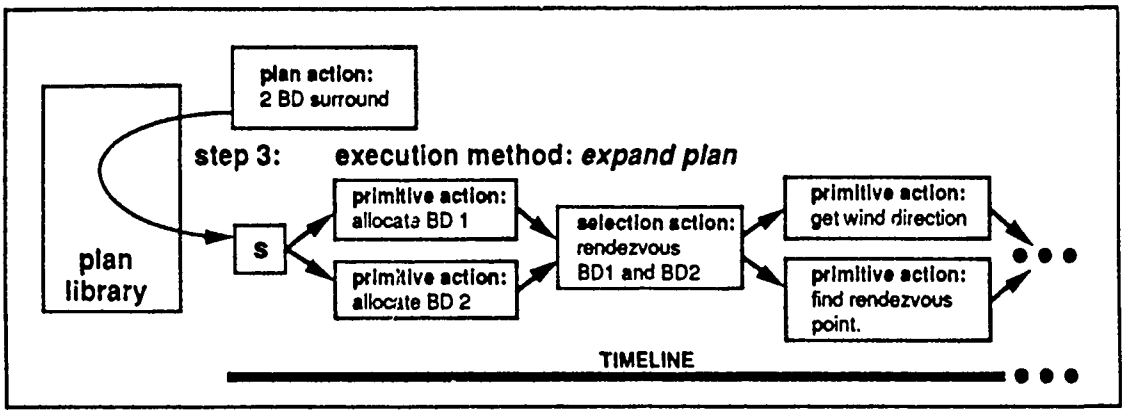

Figure 4: The fireboss executes timeline action, 2 BD surround, which expands into a network of plan steps

must be selected to go first. Then, as before, the cognitive scheduler assesses the available time and selects an execution method.

Three kinds of actions can be differentiated by their effects on the timeline when they are executed: *selection actions*, like "deal with fire 27" result in a search of the plan library, after which a plan action such as "2 BD surround" is posted on the timeline. *Plan* actions are placeholders for plans; executing them results in plan expansions being posted on the timeline. Many of the actions in a plan are of the third type: *primitive* actions that result in a computation (e.g., calculating a route), or a command to a sensor or effector. But a plan can contain any of the three types of actions; for example, the expansion of "2 BD surround" contains a selection action. When executed, it will result in a search of the plan library for a plan to rendezvous the two bulldozers. Plans can also contain plan actions, which, when executed, add subplans to the network. This is our mechanism for representing hierarchical plans. Lastly, plans may contain just a single, primitive action, such as finding the rendezvous point for two bulldozers.

We have discussed how actions are scheduled and executed when everything goes according to plan, but in the Phoenix environment it rarely does. Phoenix agents have three abilities, all rudimentary, to handle unexpected events. Reflexes, operating on a very short time scale, can halt or modify potentially injurious actions, such as straying into the fire. By design, reflexes do very little processing, and return

very little information. When a reflex halts a bulldozer, it simply posts a flag for the cognitive component; it does not interrupt the cognitive component to explain what it did. The cognitive component doesn't become aware of the change until it executes a regularly-scheduled status-checking action. In fact, by design, nothing ever interrupts a cognitive action. This is because the cost of saving state and switching context may be prohibitive. Instead, the reflexive component of a Phoenix agent is expected to deal with situations as they arise. Most, like staying parallel to a moving fire, will never require the attention of the cognitive component anyway; but even when a serious problem comes up, the reflexive component is designed to keep the agent functioning until the cognitive component finishes its current task.

The second mechanism for handling unexpected situations is error recovery and replanning. Errors are unexpected events that preclude completion of an action or a plan. For example, bulldozers will travel to their designated destinations but fail to find a fire, path planning will sometimes fail to generate a path, selection actions will search the plan library but fail to find a plan that satisfies all constraints, and so on. Currently, over a dozen types of error can arise in Phoenix, although we do not have plans to deal with them all yet. The error handling mechanism is to post on the timeline a "deal with error" selection action, which, when executed, generates a plan for dealing with the error. Currently, error recovery involves very little tinkering with the actions that are currently on the timeline, that is, no serious replanning.

Lastly, Phoenix agents have limited abilities to monitor their own progress. This is accomplished by generating expectations of progress, and matching to them actual progress. In the near future, this mechanism will enable Phoenix cognitive components to predict failures before they occur.

In sum, planning is accomplished by adding a selection action to the timeline to search for a plan to address some conditions. Executing the selection action places an appropriate plan action or primitive action on the timeline. If this new entry is a plan action, then when it is executed, it expands into a plan by putting its subactions onto the timeline with their temporal inter-relationships. If it is a primitive action, execution instantiates the requisite variables, selects an execution method, and executes it. In general, a cognitive agent will interleave actions from the several plans it is working on.

This style of planning is "lazy skeletal refinement"—lazy because some decisions are deferred until execution time. Specifically, plans are not selected until selection actions are executed, and execution methods are selected only when an action is about to execute. This style of planning and acting is designed to be responsive to a complex dynamic world by postponing decisions, while also grounding potential actions in a framework (a skeletal plan) that accounts for data, temporal and resource interactions. The combination of a reflexive and cognitive component is designed to handle time scale mismatches inherent in an environment that requires micro

actions (e.g., following a road) and contemplative processing such as route p'nning, which involves long search times and integration of disparate data. We must stress, however, that Phoenix is too early in its development to claim that our agent design is *necessarily* the best one for the Phoenix environment.

5.3.3 The Organization of Fire-Fighting Agents in Phoenix

The fourth level of the Phoenix system is currently a centralized, hierarchical organization of fire fighting agents. Because all agents have the same architecture, many other organizations of agents are possible. Our centralized model is neither robust (e.g., what happens if the fireboss is disabled?) nor particularly sophisticated. But it is simple, a great advantage in these initial phases of the project. One fireboss coordinates all fire fighting agents' activities, sending action directives and receiving status reports, including fire sightings, position updates, and actions completed. The fireboss maintains a global view of the fire situation based on these reports, using it to choose global plans from its plan library. It communicates the actions in these plans to its agents, which then select plans from their own plan libraries to effect the specified actions. Once their plans are set in motion, agents report progress to the fireboss, from which the execution of global plans is monitored. All communication in this centralized implementation is between the fireboss and individual agents - there is no cross-talk among the agents.

The fireboss maintains global coherence, coordinating the available fire fighting resources to effectively control the fire. It is responsible for all the work required to coordinate agents, such as calculating rendezvous points, deciding how to deploy available resources, and noticing when the fire is completely encircled with fireline. The plans in its plan library are indexed by global factors, such as the size of the fire and the weather conditions. The actions in its plans are mostly concerned with coordinating and directing other agents. The fireboss' state memory records the current environmental conditions, where agents have seen fire, what actions have been taken, what agents are available, and how well global plans are progressing. The fireboss is currently implemented without any sensors, effectors, or reflexes. It a cognitive agent that relies solely on communication for its knowledge of what develops in the outside world, although it does have a map of the static features of Yellowstone.

Each of the other fire fighting agents has a local view of the environment based on its own sensory input. They have access to maps of the static features in Yellowstone such as ground cover, roads, and rivers, but only know about dynamic processes such as the fire from what they see or are told by the fireboss. Sensors have a limited radius of view, though agents are able to remember what has been perceived but is no longer in view. The fireboss's global view is available to an agent only through communication. A bulldozer is an example of an agent type. It has a movement

effector that can follow roads or travel cross-country. When it lowers its blade while moving, it digs fireline and moves more slowly. It has a sensor that sees fire within a radius of 512 meters. Another sensor picks up the contour of a fire (within its radius of view). When a bulldozer is building fireline at the contour, it uses the follow-fire sensor in combination with the movement effector (with lowered blade) and a reflexive action that helps maintain a course parallel to the contour. As the contour changes, the contour sensor registers the change, which triggers a reflex to adjust the movement effector's course. The bulldozer's plan library has plans for simple bulldozer tasks such as following a given path or encircling a fire with fireline.

Although all agents have the same architecture (i.e., timeline, cognitive scheduler, plan library, state memory, sensors, effectors, and reflexes) they do not have the same plans, reflexes, sensors or effectors. The difference between the fireboss and other agents lies in their views of the world and the types of plans each knows. The lines of authority and division of responsibilities are clear: The fireboss maintains the global picture, based on the local views of its agents, and it executes plans whose effects are to gather information, send directives to agents, and coordinate their activity via communications. In contrast, the agents execute plans whose actions program sensors and effectors, which in turn effect physical actions in the world. In some sense the fireboss is a "meta-agent" whose sensors and effectors are other agents.

5.4 Scheduling the Timeline

Since the system may be dealing with multiple fires concurrently, the timeline may have many actions that are "eligible for execution" at any given point in time. As has been mentioned, the cognitive scheduler decides:

- which action to do next
- how much time should be allocated to its execution
- what execution method should be used to implement the action

This section deals with mechanisms used to decide which action to do next. It should be noted that the strategies described in this section have been formulated as initial strategies for dealing with these problems. We expect that as we gain experience with the allocation problems that are encountered, these strategies will undergo significant revision.

This section of the report has three major components. In the first, we outline the major parameters that must be specified in designing a time management strategy. We also mention the kinds of information the cognitive scheduler must have in order to make an appropriate scheduling decision. We then mention some relatively traditional

scheduling strategies that can be applicable in this type of system and give some indication as to how they might be applied. An alternative approach to scheduling that blends traditional mechanisms with qualitative reasoning about time allocation is then outlined. Finally, we discuss a model of time that views time as a resource. Viewing time from this perspective gives us a basis upon which to build algorithms for implementing a hybrid time management scheme.

At the most abstract level, the decision as to what to do next is always based on some scheduling strategy. The scheduling strategy used biases the scheduling decisions and perhaps the methods used to implement basic planning tasks (e.g., selecting and instantiating plans). Thus the scheduling strategy should be adaptive so that it can respond to the needs of a planner operating in a dynamic environment. For example, when the planner is running far behind schedule, a *panic mode* strategy might be appropriate so that only the most important actions get scheduled and only the quickest methods for doing them are selected. On the other hand, when the situation is less time-critical, a scheduling strategy that is able to construct more nearly optimal time allocations should be used.

At any given time, the timeline contains all the cognitive actions the agent would like to perform. In addition, the timeline contains precedence constraints among the mental actions and deadlines for some actions. The scheduling problem can be stated as follows:

> Given the timeline and information about the real-time nature of each mental action, decide which mental action to perform next, and how much time to allocate to that action.

At any given time, there is a current scheduling strategy in effect. This strategy is selected within the top level loop of the cognitive scheduler. Methods for choosing which action to do next are provided with the strategy.

In the paragraphs which follow, we use F to denote the function that decides which action to do next and the amount of time to allocate to that action. Clearly, F is defined by the current scheduling strategy. As has been mentioned, one possible scheduling strategy involves "panic-mode" activity. When in panic-mode, time is at a premium, so all decisions, including scheduling decisions, must be made as quickly as possible. Thus F would simply return the first action it finds that can be executed, and allocates the minimum time necessary to that action. A more typical strategy may be a "minimum-slack-time-first" (MSTF) strategy. MSTF calculates the slack time available for each action in the timeline. F then returns the action with the smallest slack time. Slack time can be allocated to actions based on some priority scheme, or saved for the future when there may be a higher demand for computes.

It is important to recognize that there may be a high start-up cost associated with any change in scheduling strategies. Scheduling algorithms often require particular

data structures to maintain scheduling information and gathering that information may require searching the entire timeline. Consequently, it is not likely that scheduling strategies will be switched frequently.

In order to completely specify a scheduling strategy, each of the following parameters must be clearly defined:

context/applicability: When is this strategy appropriate? For example, "lots to do in a short time."

initialization-function: Some initialization of the scheduler's global state is required for each scheduling strategy. For example, with an MSTF strategy, a initial pass through the timeline must be done to calculate slack times.

Scheduling function, F: Given the timeline and the scheduler's global state, calculate which action to do next and how much to allocate to it.

update-function: Any changes to the timeline made by the selected actions must be processed. In the case of MSTF, for example, this function would update slack times.

monitor?: This describes how to decide if it is time to try another scheduling strategy.

The scheduler has several sources and types of information available to assist in the decision making process. Among these are:

Precedence constraints: The timeline contains information about the order in which mental actions must be done.

Deadlines: The timeline contains information about desired start and end times for mental actions. Since the deadlines may be soft, a more detailed description of the deadline may be given. Is the deadline strict (start/end at a specific time) or relative (start/end after/before a time)? How hard is the constraint? Is it OK to be a little late?

Durations: Associated with each action (or method for performing an action) is a set of durations.

Priority: How important is each action? This may be used, for example, to allocate slack time in MSTF.

Atomic/Preemptable: Some actions can be preempted and resumed later; others cannot.

Local scheduling preferences: Actions may have specific scheduling preferences. For instance, some actions would like to be done as late as possible.

5.4.1 "Traditional" Scheduling Strategies

Some of the potentially useful scheduling strategies can be adapted from techniques that have been developed in other contexts. Examples of some of these are given below.

Panic-mode: Panic-mode scheduling is applicable when there is mental overload. In this situation, we want to spend as little time thinking about scheduling as possible. Panic-mode allocates as little time as possible to the task with the highest priority. (Of course, this may be exactly the wrong strategy to apply. It may be necessary to do careful reasoning about the timeline to decide exactly which tasks can be safely delayed and which ones cannot. On the other hand, we do not want the system thrashing in the scheduler).

Minimum Slack First: Minimum slack time first is based on one of the set of scheduling algorithms that come from a traditional PERT/CPM analysis. When using MSTF, the scheduler does a forward and backwards pass through the timeline to calculate the earliest start time (EST) and latest start time (LST) for each mental activity. By definition, slack time (ST) can be calculated as $ST = EST - LST - duration$. When the slack time for any task is negative, it is impossible to come up with a schedule that meets all its deadlines. Assuming the schedule is feasible (positive slack time for all tasks), this scheduling strategy executes the action with the minimum slack time first. The amount of time allocated to the task is a function of the time required by the task and the amount of slack time in the system. Since tasks with small slack time are more sensitive to the uncertainties in the "mental time domain", they are executed first.

There are some problems with this approach. First, when the knowledge about deadlines is bad (uncertain), measures of slack time are not very meaningful. Second, there is no obvious definition of slack time when activities cannot be done in parallel. The slack time depends on the order that actions are executed. Finally, what do we do when the schedule is infeasible? Negative values of slack time tell us how late an action will be.

Earliest Latest Finish Time First: ELSTF is another PERT/CPM algorithm. In this case we calculate the latest time each action can conceivably be finished and execute the action whose deadline is earliest first. (Theoretically speaking, when there is a feasible schedule, ELSTF can generate schedules that violate deadlines whereas MSTF always guarantees feasibility). This approach is subject to the same problems as MSTF. ELSTF requires about half as much time to execute as does MSTF.

5.4.2 An Alternative Approach

An alternative approach to scheduling the timeline involves a hybrid scheduling strategy that blends qualitative reasoning about time allocation with more traditional scheduling mechanisms. Such an approach employs flexible decision making and control strategies that should not incur the high degree of overhead associated with changes of scheduling strategy that has been mentioned previously.

This approach to scheduling assumes (as do the other strategies that have been discussed) that actions on the timeline have estimates of computation time and execution time associated with them as well as an assessment of their criticality. This strategy differs from others in that each action type also has a set of heuristic rules associated with it. These rules are used to assess the degree to which it is advisable to schedule an action. They reflect qualitative factors associated with each action, as opposed to such metrics as slack time, which are quantitative. When triggered, an assessment rule places a positive or negative endorsement [3] on some action being considered by the scheduler.

The fundamental scheduling strategy can be summarized as follows:

1. Endorse actions on the timeline with the aid of heuristic assessment rules.
2. Perform a cost/benefit analysis based on strength of endorsement, criticality of action, computation time required, and execution time required.
3. Rank actions based on the cost/benefit analysis.
4. Schedule on the basis of the ranking, consistent with the partial order of the timeline.

There are some obvious difficulties with a strategy such as this. The most significant of these difficulties is that there may not be sufficient time available to the scheduler to perform thorough analysis relative to deriving endorsements. To obviate some of these difficulties, the use of progressive reasoning [6] is incorporated.

If there is an imminent emergency, the scheduler has no time to "think" and all analysis relative to endorsement is bypassed. Scheduling is performed based only on the assessment of criticality associated with each action (and, of course, the partial order inherent in the timeline).

For cases in which there is time for the scheduler to think, we have identified three levels of reasoning that appear to be relevant. At the first level, all rules that can be applied without requesting additional (or more recent) information or performing computations are triggered, and a first level assessment is made. On the second level, a "request list" is formed. This request list reflects information that would help the

scheduler arrive at a more informed (hence better) schedule. This level is concerned with updating old information and/or verifying parameters on the request list. A (more informed) assessment is made at the end of this level of reasoning. Finally, in the third level, parameters perceived to be inaccurate or less accurate than desired are recomputed, and a final assessment is made. If the scheduler's time to think expires during a particular level of reasoning, the ranking determined at the previous level is returned.

As has been mentioned, our hybrid time management strategy assumes that its input consists of a set of proposed actions, each of which has associated with it an estimate of its time requirements as well as any relevant deadlines. There may be temporal precedence relationships among some of these proposed actions. The task faced by the time manager is one of allocating time among the proposed actions in such a way that adaptive, time sensitive global problem solving is promoted.

We assume that the proposed actions fall into a fixed, domain specific set of *action types*. In addition, for each action type, there is a set of (domain specific) asssessment rules to be used in evaluating the importance and benefit of performing actions of that type. Preconditions in these rules deal with dynamic, time varying domain characteristics and are associated with *criticality factors* to be used in progressive reasoning by our time management strategy. As the result of rule firing, qualitative assessments of utility are associated with the relevant actions. These assessments are dependent on those preconditions that triggered the rule firing.

The overall hybrid time allocation algorithm is structured as follows:

1. Respond to emergency situations immediately, without delay.
2. If there are urgent (not emergency) situations, apply knowledge-poor strategies to allocate time for them.
3. If there are no emergencies and available deadlines indicate that thinking about time allocation is in order:
   (a) Determine how much time to allocate this cycle, based on available deadlines and the degree of dynamic change in the environment.
   (b) Set k to the maximal criticality factor present.
   (c) While there is still time for thinking about time allocation:
      i. Trigger any eligible assessment rules based on preconditions with at least the current criticality factor;
      ii. Decrement the current criticality factor considered;
   (d) Perform cost/benefit analysis to determine relative utility of actions.
   (e) Allocate time proportional to perceived utility.

During the last year, we have focused our attention largely on that part of the overall control loop associated with step 3 above. The criticality factors associated with preconditions are reminiscent of the ABSTRIPS criticality factors [5]. Rather than using criticality factors to yield increasingly detailed plans, though, we use them to give us increasingly more knowledgable assessments of the utility of performing a given action. The intuition behind the mechanism depends on the observation that humans often make decisions based on partial information (when there is no time to gain more complete knowledge). As time permits, human problem solvers make assessments that are based on an increasingly complete and accurate picture of the problem state. Thus we reason until there is no more time to reason then make decisions based on the most recently determined evaluations (which are presumably those that are the most informed). This model of the human decision making process is reflected in the algorithmic strategy outlined above. Use of progressive reasoning in this way extracts a spectrum of decision making activity ranging from a purely reactive mode in time critical situations to a reflective mode when there is time to think.

Progressive reasoning applied in this context should embody the reactive-reflective spectrum of decision-making employed by the cognitive scheduler. It is interesting to note that the planning and monitoring tasks become interwoven in the second and third levels of reasoning. For example, projections about the extent of the fire in a certain area, or updates of the front of the fire, may be requested by the scheduler during these levels, as the scheduler may need this data to make a more informed decision. The problem we see is to characterize and take advantage of this interaction rather than attempting to avoid it.

5.4.3 Viewing Time as a Resource

In order to gain perspective as to how time can be allocated using this hybrid type of strategy, we view time as a resource, and consider the scheduling problem as one of allocating a scarce resource. In this section we identiiy three different types of "time resources", discuss some of their properties, and describe interactions among them. We then mention a number of constraints arising in this resource allocation problem, several sources of contention for resources, and finally, discuss some heuristics for allocating time.

As has been mentioned, we define three types of time: act-time, think-time, and idle-time. Loosely speaking, act-time corresponds to the time it takes to perform a "real world" action, think-time is computation time, and idle-time is characterized as the absence of either think-time or act-time (depending on the subtype). We further assume that there are three "reasoning agents", the scheduler (S), the planner (P) and a monitoring agent (M). The system also has some number of "effector agents," agent1, ..., agent$k$ for some k. These effector agents represent the agents in the field

performing actions to control the fire. Examples include bulldozers, crew, helicopters, and planes. The "reasoning agents" consume think-time and idle-time; the "effector agents" consume act-time and idle-time. (For the time being, we assume that effectors are "brainless", acting only at the behest of the reasoning agents.) With respect to time allocation, our hybrid scheduling strategy has the following goals: to maximize useful "effector agent" actions, taking advantage of as much parallelism as possible, to maximize the usefulness and efficiency of CPU usage, and to minimize all idle-time.

Suppose that we are given an interval of time, (t, T). This gives rise to a "pool" of time resources available, for some value of t and T. This pool (call it R) can be viewed in two different ways. In one sense, R can be viewed as an interval divided lengthwise into two bands, one labeled "think-time" and one labeled "act-time." The act-time band is further divided lengthwise into k bands, one for each effector agent. When all think- and act- time has been allocated, these bands will have been divided into resources, much as an interval can be divided into subintervals. In agent*i*'s act-time band, the division would correspond to the time agent*i* takes to complete any actions it performs within (t, T), as well as the position within (t, T) in which they were performed. Any "gaps" would correspond to idle-time. Initially, before any resources are allocated, the bands are undivided.

We may also consider R as a collection of sets of possible resources to allocate, only one element of which is ultimately realized when the available time interval has been allocated. This is evident when we consider an example. Clearly, act-time and think-time are measured in different units. It takes linger for a bulldozer to *get* to a location than it does to decide to send it there. For the sake of illustration, suppose act-time is measured in 2-second intervals, and think-time in 1-second intervals. Further assume that the total length of the interval (t, T) is 6 seconds. Then think-time may be divided among the reasoning components as follows: 1(S) + 1(M) + 1(P) + 3(idle), which means that the scheduler has 1 second of CPU time, then the monitor, then the planner, followed by a three second interval when the CPU is idle. The same 6 second interval could also have been allocated as 1(M) + 2(S) + 3(idle), or as 1(S) + 1(M) + 1(P) + 1(M) +2(P). In order to consider all possibilities, one must consider all partitions of 6, along with all permutations allowing for different orderings of usage, as ,has been partially illustrated. Using the same example scenario, agent1's act-time could be divided up as follows: action1 takes 4 seconds, action2 takes 2 seconds; or agent1 is idle for 2 seconds, action1 takes 4 seconds, etc. To obtain one set of possible resources in R, we take one particular allocation for think-time, together with one allocation for each effector agent's act-time. All possible such sets together comprise R.

The problem of optimally allocating time (when it is viewed in this manner) is clearly intractable. With this view of time, the process of allocating time can be viewed as one of progressively eliminating sets from R, with the goal of having an

"optimal" set remain as the single set finally left in R. Heuristics that reduce the remaining alternatives in R very quickly are necessary.

In the following paragraphs, we summarize various observations that we have made concerning properties of time (when viewed as a resource to be allocated among various tasks). We anticipate that these observations will result in further refinement of our model of time and in development of heuristics for effective time allocation.

5.4.3.1 Characterizations of Time Resources Time, when viewed as a resource, has a number of significant attributes. Some of these attributes are common to all "types" of time, while others are specific to one kind of time. Various attributes of importance in time allocation are mentioned in this section.

Act-time

1. consumed by "effector agents"
2. allocated by the scheduler
3. a specific instance of allocated act-time has 3 properties:
   (a) length of duration
   (b) position within a time interval (t, T)
   (c) "effector agent" associated with it
4. for a given "effector agent" an instance of act-time, once consumed, cannot be re-used
5. within any given subinterval of (t, T), there may be several instances of act-time currently being consumed (by different agents)
6. measured in units of r(act), probably given in units of minutes to hours
7. can be consumed concurrently with think-time (though there are some restrictions, which will be noted later)
8. purpose is to effect an action; at the end of an instance of act-time, the agent using it has performed an action

Think-time

1. consumed by "reasoning agents", the scheduler, monitor, and planner
2. allocated by the scheduler

3. a specific instance of allocated think-time has two properties:
   (a) length of duration
   (b) position in (t, T)
4. a specific instance of think-time, once consumed, cannot be re-used
5. within any given subinterval of (t, T), there may be sequential but not concurrent instances of think-time being consumed
6. measured in units of r(think), probably given in units of seconds to minutes
7. can be consumed concurrently with act-time
8. purpose of think-time is decision making and computation

Idle-time

1. there are 3 types of idle-time:
   (a) (think) CPU is not in use
   (b) (act) a particular agent is not acting
   (c) (act) no agent is acting
2. idle-time is not allocated specifically, but is defined as the absence of either think-time or act-time being consumed
3. a specific instance of idle-time has 2 properties, the precise characteristics of which are determined by the allocation of think-time and act-time:
   (a) length of duration
   (b) position within (t, T)
4. idle-time of the "act variety" has an "effector agent" associated with it
5. measured in units of either r(act) or r(think), as appropriate

5.4.3.2 Interactions Between Types of Time It is clear that act time, think time, and idle time all share some attributes, while each type of time has some features that distinguish it from the others. It is also possible to identify several forms of interaction among these types of time.

1. All act-time instances must be preceded by some think-time instances. Thus, an act-time instance must be preceded by some scheduler think-time and possibly some planner or monitor think-time specific to that action.

2. Some think-time instances may be preceded by an act-time instance. This could happen, for example, if the scheduler needed more information for decision-making. (This constitutes an example of the interweaving of monitoring and scheduling mentioned previously).

3. Some act-time may be allocated within scheduling.

4. Idle-time's precise characteristics are determined by the those of the surrounding think-time or act-time.

5. Think-time and act-time may be consumed concurrently, when thinking is relative to implementing future actions.

Since the ratio r(act)/r(think) is often large, there are two consequences of observations (2) and (3). First, act-time should not be allocated within scheduling unless the total amount of time allocated to the scheduler is large, and can be measured in terms of r(act). Secondly, should this kind of act-time be allocated, the scheduler would issue a request for information and could not continue on its current line of thought until that information is provided. Thus there is the potential for existence of a large piece of idle-time in the think-time band. This could result in a significant loss of usable computation time. Measures to avoid this type of scenario must be devised.

5.4.3.3 Contention for Time Resources Time is a scarce resource in real time environments. Agents compete for computation time, as do tasks within an agent. This section summarizes a number of observations concerning the nature of contention for time.

Act-time

1. Inter-agent

   (a) there is no contention for act-time between different "effector agents" for independent, non-mutually exclusive actions

   (b) for mutually exclusive actions: only one of the agents may be allocated act-time

   (c) if partial ordering of actions is known in advance, the allocated act-time position property must behave accordingly

2. Within an agent

   (a) for mutually exclusive actions, effect of allocated act-time must be considered in advance, and act-time must be allocated with that in mind

Think-time

1. Sources of contention for think-time:
   (a) scheduler for decision-making
   (b) planner for determining details of implementing an action
   (c) monitor for determining details of implementing an action
   (d) monitor for computations, such as projections of fire, etc.
   (e) planner and monitor for replanning, enable a new decision-making cycle
   (f) updating a database describing the state of the fire
2. None of the above can be done concurrently.
3. High level of contention for think-time resources. All activities mentioned in (1) are necessary.

Idle-time

1. no contention; nobody wants idle-time!

5.4.3.4 Constraints on Time Allocation Many of the constraints on allocating different types of time resources have already been mentioned. In particular, interactions between instances of time give rise to constraints, as do certain aspects of contention among resources. To some extent, these constraints may already be reflected in the timeline.

As already noted, an act-time instance must be preceded by its corresponding think-time, as well as scheduler decision-making time. If there is a deadline by which the action must be completed, this constrains:

- amount of scheduler decision-making time
- allocation of think-time relative to the action
- allocation of think- and act- time relative to other actions

If there is no such constraint on an action, the "beginning point" of its act-time is constrained to be after the "end point" of its think-time (in "wall time").

In addition to the difference in scaling, think-time and act-time have another fundamental difference that can be seen as a kind of constraint. Once an instance of think-time is allocated, there is no potential for extending the total length of that instance. For example, if the scheduler is allocated 5 seconds of think-time, beginning

at a particular point in "wall time," then after those 5 seconds are consumed, control of the CPU is granted to the next "reasoning agent." However, if bulldozer1 is given 10 minutes to get from PointA to PointB, and it takes 12 minutes to do so, in some cases this is fine, and bulldozer1 is allowed the full 12 minutes to complete the action. This means that there is less control over an "effector agent" consuming precisely the resource it is given than with a "reasoning agent."

5.4.3.5 Criteria for Allocating Time as a Resource Heuristics are clearly needed in allocating time well. An approach to scheduling that explicitly considers time as a resource attempts to blend qualitative reasoning with more traditional scheduling heuristics. In this section, we mention some of the parameters that affect these heuristics.

Broadly speaking, there are two types of actions considered in this type of system: one that brings about an "effect" in the "real world" and one that increases knowledge. In addition, actions may have originally been placed on the timeline and they may reflect scheduler requests for information.

Factors affecting time allocation for actions placed on the timeline:

- Contextual reasons for wanting the action to be performed:

  Does the situation in question merit this action, and how strongly? Are there reasons that the action is warranted, considering what we know (or think we know) about the future? These kinds of concerns are captured in the rules of our system, and through the use of endorsements.

- Other factors to consider:

  Is there a time constraint on when action must be completed? If not, will the CPU be tied up too long before this action is given a chance? How much think-time is involved in this action? These concerns are incorporated in specific efforts to rank actions, and to consider time itself as a cost in the cost/benefits analysis.

Factors associated with allocating act-time during scheduling:

- Length of allotment of act-time should not be "too large" (so determine an upper bound).

- Can the concurrent think-time be used constructively? We do not want the CPU to be idle for long periods of time.

- If a request for information is issued by the scheduler and n time units are allotted, the scheduler will wait no longer than n time units for that result.

In this case, the scheduler is more likely to want to allocate this time if it is reasonably certain it will get a result within the time allotted.

It is clear that during each scheduling cycle, the scheduler needs at least some minimal time to think, if only to decide to sidestep the full-blown process. Similarly, the global database should be updated on a regular basis. The planner and monitoring functions need some amount of time for replanning purposes, so the amount of time given to these activities must be determined. Not all actions currently listed on the timeline will necessarily be allocated CPU time. The position in "wall time" and length of an instance of think-time will, to some extent, be decided by the amount of time requested. For example, an action that takes very little CPU time would tend to have higher priority than something that is very computation intensive.

Algorithms that implement a time management strategy based on this model have been designed and implemented. Experimental data concerning the behavior of these algorithms under typical scenarios found in the firefighting domain is being collected.

5.5 The Multiple Fireboss Scenario

The bulk of the work that has been done in the firefighting domain to date has assumed that there is one fireboss whose job is to direct activity relative to containment of the fire. This would imply that there is a single high level planning agent whose responsibility is to (centrally) generate a plan for fire containment.

We have also been investigating issues that arise when we admit the possibility of multiple firebosses, each of which has primary responsibility for firefighting activity in some geographic region. Under this set of operating conditions, the firebosses each have only limited knowledge about the state of the fire outside their region of responsibility and each may have limited communication bandwidth available to ascertain the status of a fire that extends beyond its own regional boundaries. Environments involving multiple firebosses clearly require a different kind of planning strategy than those in which there is a single overall fireboss.

Construction of a multiple fireboss planning environment requires that several issues be addressed. Among them are the communication protocol that is utilized by the firebosses, mechanisms for determining when cooperation among firebosses is advisable, and mechanisms to be utilized by a fireboss in determining the form of assistance that it is willing to offer a neighboring fireboss in the event that aid is sought.

We presently have formulated preliminary strategies for handling multiple fireboss scenarios. The current method of evaluation dictates that a fireboss use only a fire's location as a factor in determining whether to request aid. If the fire is located near a regional boundary, aid is sought from the appropriate neighboring fireboss. Each

neighbor fireboss responds to a request for aid by indicating that it cannot help out ("No") or that it can be of some assistance ("Yes"). An affirmative answer is based only on the number of bulldozers available. If an affirmative response is sent, the number of bulldozers that can be contributed to the effort is also indicated.

It is clear that a decision to respond positively to a neighbor's request for aid requires that a fireboss agent be able to assess its own status relative to that request in a reasonable fashion. It must be able to determine what the costs (to itself, in terms of resource it will not have available) of sending aid. These costs may be measured in terms of its own reduced firefighting capacity, in terms of the likelihood of fire occurring it in its region, and in terms of the costs associated with relocating its own bulldozers (i.e. the time it would take to reposition them if need be).

An agent must also be able to assess the potential benefits of sending aid to a neighbor. In some cases, sending aid to a neighbor in a timely fashion could serve to contain the fire before it spreads in to this fireboss's region of responsibility.

5.6 Status and Concluding Remarks

At the outset, we mentioned that the research efforts on this task have been concentrated on formulation and implementation of an appropriate agent model and mechanisms for handling time in general and reasoning strategies fo radjudicating allocation of time among various cognitive activities in the planner. An appropriate agent model has been formulated and incorporated in a (centralized) planner for the firefighting domain [4].

We have also seen that time is a critical resource for planning in this type of environment. When time is viewed as a resource, proper allocation of time among subtasks is critical in achieving reasonable performance. We have noted that one problem central to development of heuristics for determining time allocations is that of formulating ways of handling the fact that time can be viewed in more than one way.

On reflection, it seems evident that the CPU time associated with the planner is measured in seconds to minutes, whereas the time associated with acquiring information regarding the state of the fire may be measured in minutes or hours. Reasoning about actions in an environment such as this requires that the planner understand and be able to deal with these extreme differences in scale. Effectively, two types of time: "execution time" and "action time" or "internal time" and "external time". These two types of time share some attributes, but are fundamentally different in others (as far as the planner is concerned).

Research concerning a model of distributed planning, agent characteristics, and effective ways of modeling time has been initiated and mechanisms for experimenting

with time (as perceived by the agents in the system) have been incorporated in the simulator design. In addition, qualitative reasoning about time allocation has been investigated and appropriate algorithms have been implemented. Finally, an investigation of ways in which multiple "firebosses" can be accommodated has been initiated. Activity continues relative to all of these problems.

References

[1] P. E. Agre and D. Chapman, "Pengi: An Implementation of a Theory of Activity", **Proceedings of the Sixth National Conference on Artificial Intelligence**, August 1987, pp. 268-272.

[2] R. A. Brooks, "A Robust, Layered Control System for a Mobile Robot", **IEEE Journal of Robotics and Automation, RA-2 (1)**, 1986, pp. 14-23.

[3] P.R. Cohen, **Heuristic Reasoning about Uncertainty: An Artificial Intelligence Approach**, Pittman Publishing, Inc., 1985.

[4] P. R. Cohen, M. L. Greenberg, D. M. Hart, and A. E: Howe, "Trial by Fire: Understanding the Design Requirements for Agents in Complex Environments", **AI Magazine**, Volume 10, Number 3 (Fall 1989), pp. 34-48.

[5] E. Sacerdoti, "Planning in a hierarchy of Abstraction Spaces", **Artificial Intelligence 5**, pp. 115-135.

[6] M. L. Wright, M. Green, G. Fiegl, and P. Cross, "An Expert System for Real Time Control", **IEEE Software**, March 1986, pp. 16-24.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic reliability/maintainability and compatibility.